The Com-plete Task Structure

When a program is to be initialized, a Com-plete Unit of Work Control Block (CUOW) is allocated and checks are made to insure that the user is allowed to run the particular program. Once this has been done, the work is dispatched as described in this chapter.

This chapter covers the following topics:

- Dispatching (Task Selection)
- Thread Selection and Reservation
- Relocation
- The Quiesced State

Dispatching (Task Selection)

Task selection must first determine which task group to use for the work. If no task group is explicitly assigned to the program in its catalog entry, the DEFAULT task group will be used. Dispatching is achieved by ENQuing and DEQuing the Terminal Information Block (TIB) from queue to queue. The Processor Group Control Block (PGCB) has four queues associated with it, one for each possible TIB priority. The TIB is ENQued to the appropriate PGCB queue. If the system is active, it is likely that the TIB will be selected from a queue by one of the active tasks. If this is not the case, the first available task is found by searching the chain of Processor Control Blocks (PRCB) and posting its appropriate task active. The PRCB running the unit of work reserves the required thread and runs the work.

There are cases where the use of certain system functions can cause a program to have affinities to certain operating system tasks. When this occurs, the work is ENQued directly to a PRCB related queue for the task in question. If and when the condition for the affinity no longer exists, the TIB can once again be ENQued to the PGCB queue where it can be processed by any of the tasks in the task group. Work with an affinity is totally dependent on one task and must wait for that task to complete any other work it might be doing. For this reason, affinities should be avoided at all costs.

For OS type systems currently, the only reason a task will have an affinity to a task is when that task issues an OPEN. This is due to the fact that the equivalent CLOSE must be issued on the same operating system task. It is also possible to indicate in a program’s catalog entry that it must run with affinity where it would not be possible for Com-plete to internally detect that this is necessary.

Thread Selection and Reservation

Thread selection first involves finding the thread group within which the program will run. If no thread group is explicitly assigned in the program’s catalog entry, the DEFAULT thread group will be used. Once the thread group has been found, an eligible thread sub-group must be selected for the work. The first sub-group found that provides the required amount of storage below the line will be the sub-group where the work will run. The sub-groups are searched from the one with the smallest amount available below the line to the one with the largest amount available. Once the sub-group is selected, an attempt will be made to reserve a free thread within that sub-group. If this is successful, the program starts executing. Otherwise, the program is queued to the thread sub-group and will start executing as soon as a thread of this sub-group becomes available.

While competition for threads should be avoided where possible by allocating sufficient threads in the system, it can still occur. When a non-relocatible program is about to continue its execution after it was rolled out of its thread, it must be rolled back into the same thread. For this reason, once a unit of work has been selected by a task, it must reserve the thread where the work will be carried out. If the thread is busy, the work will be queued to the thread and will only be carried out when the previous work in the thread has finished.

In this case, the task which originally dequeued the TIB is free to go and do other work once the CUOW has been queued to the thread. If the CUOW in question has an affinity to the task, it must not only wait for the thread to become free, but also for it’s associated task to become free again at the same time. In a busy system, programs running with affinity could suffer relatively long delays waiting for both its thread and task to be available.

Relocation

Relocation is only important if there are an insufficient number of threads to service the maximum number of users who will be concurrently active. The logic here has not changed in that the thread will be prepared for relocation on rollout and the connection with the thread broken. When they are to be rolled in again, the thread selection takes place in the same way as described earlier.

The Quiesced State

The primary purpose of the QUIESCE state is to enable users currently working to finish what they are doing in an orderly fashion but preventing new work from being started. When the system has been quiesced using the QUIESCE operator command, the following will occur:

VTAM will accept no more requests to start sessions with the Com-plete in question. VTAM sessions started before the QUIESCE command was issued will be unaffected.

Attempts to start new sessions via access, be they from batch or some other source will be rejected with an Adabas response code 148 to indicate that the requested node is not active. Access sessions started before the QUIESCE command was issued will be unaffected.

Users already logged on can continue working. However, they will receive a warning message each time they return to their COM-PASS-pass menu indicating that the system is quiescing and that they should finish their work and log off

Once a QUIESCE has occurred, Com-plete must be terminated before normal service can be resumed.